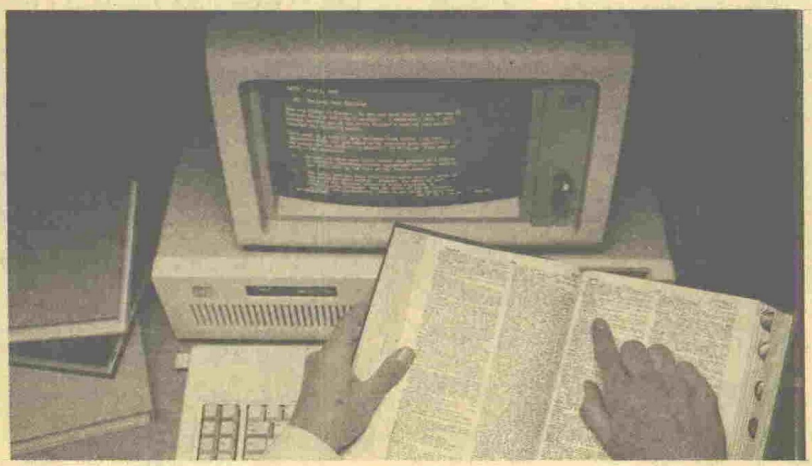
# ศึกษาภาษาสมอลทอล์ค : ออปเจ็คกับสื่อความ



บุญเลิศ เอี่ยมทัศนา

ในการศึกษาภาษาสมอลทอล์คสำหรับผู้ที่รู้ภาษาปาสคาลมาก่อนแล้ว ผมว่ามีเรื่องใหม่อยู่เพียง 4 เรื่องคือ ออปเจ็ค สื่อความ สกุล และกิจกรรมเท่านั้น นอกนั้นเป็นเพียงส่วนประกอบของภาษากับเรื่องราวของ ST/V ซึ่งผมได้นำบางส่วนมากล่าวแล้วในไมโครฯ 56 (ท่านที่ยังไม่ได้อ่านกรุณาย้อนกลับไปอ่านก่อนครับ)

สำหรับในฉบับนี้ผมจะนำเรื่องของออปเจ็คกับสื่อความ และรูปแบบของโปรแกรมในภาษาสมอลทอล์คมากล่าวถึง โดยจะขอเปรียบเทียบกับภาษาปาสคาลเพราะใกล้เคียงกันมาก สำหรับท่านที่มีซอฟต์แวร์ของ ST/V แล้วขอให้ทดลองตามไปด้วย โปรแกรมส่วนมากจะอยู่ในไฟล์ chapter.3 กับ chapter.4

## ออปเจ็ค (Object)

ความจริงแล้วเราไม่สามารถแยกออปเจ็คกับสื่อความออกจากกันได้ การที่ผมตั้งหัวข้อว่าออปเจ็คก็เพื่อแสดงว่าจะเน้นหนักในเรื่องนี้เท่านั้นครับ

คำว่า ออปเจ็ค คือ Object ในสมอลทอล์ค โดยเฉพาะที่กล่าวไว้ในหนังสือคู่มือของ ST/V จะหมายถึงส่วนประกอบพื้นฐานของภาษาสมอลทอล์ค ซึ่งเทียบได้กับข้อมูลในภาษาอื่นนั่นเอง แต่มีวิธีการมองที่แตกต่างกัน เช่น หากเขียนคำสั่งว่า

```
3 + 5
```

ในภาษาปาสคาลจะเป็นคำสั่งที่สั่งเครื่องให้บวก 3 กับ 5 แต่ในภาษาสมอลทอล์ค 3 คือ ออปเจ็ค + เป็นสื่อความ สำหรับ 5 คือ สร้อยของสื่อความ ซึ่งสื่อความเทียบได้กับคำสั่งที่สั่งแก่ออปเจ็ค (ไม่ใช่สั่งเครื่อง) นั่นคือเป็นการสั่งแก่ 3 ให้นำ 5 มาบวกเข้ากับตัวของมันเอง

เรื่องนี้ดูก็เป็นธรรมดา แต่ถือว่าเป็นเรื่องสำคัญเรื่องหนึ่งในการเขียนโปรแกรมในแบบโอโอพี ซึ่งจะต้องมองความสัมพันธ์ระหว่างออปเจ็คกับสื่อความในที่กล่าวมานี้ครับ

สำหรับผลการดำเนินการ ในภาษาปาสคาลทำไม่ได้เพราะไม่ใช่โปรแกรม หรือแม้แต่ถ้อยแถลงที่สมบูรณ์ แต่ในสมอลทอล์คทำได้ด้วย show it

## แบบของออปเจ็ค

เช่นเดียวกับในภาษาคอมพิวเตอร์ทั้งหลายที่มีแบบของข้อมูล ในภาษาสมอลทอล์คก็มีแบบของออปเจ็คซึ่งจะขึ้นอยู่กับคลาส (class) แต่มีส่วนหนึ่งที่อาจถือได้ว่าเป็นออปเจ็คพื้นฐาน ดังแสดงในรูปที่ 1 กล่าวคือ

**เลขจำนวน** ตามรูปแสดงไว้เฉพาะเลขจำนวนเต็ม (integer) นอกจากนี้ก็มีเลขจำนวนจริง (float) และเลขเศษส่วน คล้ายกับในภาษาปาสคาลแต่มีขนาดใหญ่กว่ากันมาก เช่นเลขจำนวนเต็มบวกในเทอร์โบปาสคาลกำหนดค่าได้ไม่เกิน 65536 คือไม่ถึง 5 หลักดี ใน ST/V ต้องเรียกว่าไม่จำกัดค่าคือมีจำนวนหลักได้ถึง 65536 หลัก ซึ่งผมนับจำนวนไม่ถูกครับว่าเท่าไร

**อักขระ (character)** คืออักขระเพียง 1 ตัวตามรูปคือ อักษร A

**สตริง (String)** คือแถวของอักขระ ซึ่งเหมือนกันกับภาษาปาสคาล แต่ไม่จำกัดจำนวนอักขระ

**อาร์เรย์ (Array)** เช่นเดียวกับอาร์เรย์ในภาษาคอมพิวเตอร์ทั่วไป คืออาจ

```
1234                    Integer
$A                      Character
'This is a string'      String
#(1 2 3)                Array
```

**รูปที่ 1 แบบของออปเจ็คพื้นฐาน**

เป็นอาร์เรย์ของอักขระ เลขจำนวน สตริง หรืออาร์เรย์ของอาร์เรย์เอง ที่แตกต่างกับภาษาปาสคาล คืออาจเป็นอาร์เรย์ของข้อมูลต่างแบบกันได้ด้วย ดังแสดงในรูปที่ 2 คือ

- บรรทัดแรก เป็นอาร์เรย์ของสตริง
- บรรทัดที่ 2 เป็นอาร์เรย์ของสตริงผสมกับเลขจำนวน
- บรรทัดที่ 3 เป็นอาร์เรย์ของอาร์เรย์
- บรรทัดที่ 4 เป็นอาร์เรย์ผสมของเลขจำนวนเต็มกับอาร์เรย์ (ซึ่งมีลักษณะใกล้เคียงกับลิสต์ในภาษาโปรล็อก)

การดำเนินการกับอาร์เรย์ดังกล่าวมานี้มีลักษณะเช่นเดียวกับในภาษาปาสคาล คือ ใช้อินเด็กซ์ชี้ไปที่ข้อมูลแต่ละช่องโดยนับช่องแรกเป็น 1

## ตัวแปร

การกำหนดตัวแปรในภาษาสมอลทอล์คทำได้โดยง่าย คือ เพียงแต่กำหนดชื่อตัวแปรเท่านั้น ไม่ต้องกำหนดแบบข้อมูลกำกับดังในภาษาปาสคาล เพราะตัวแปรของสมอลทอล์คจะใช้กับออปเจ็คแบบใดก็ได้ตามที่กล่าวมากล่าวคือ

### ตัวแปรโกลบอล (Global Variable)

การกำหนดตัวแปรแบบโกลบอลทำโดยเพียงแต่ป้อนชื่อตัวแปรขึ้นต้นด้วยอักษรใหญ่ (นั่นคือในสมอลทอล์คอักษรเล็กและอักษรใหญ่มีความหมายต่างกัน) แล้วให้ประเมินผลด้วย show it เครื่องจะตรวจสอบว่าเป็นชื่อตัวแปรที่เคยกำหนดไว้ก่อนแล้วหรือไม่ ถ้าเป็นชื่อใหม่จะให้เราเลือกว่าจะกำหนดหรือไม่

หลังจากที่กำหนดแล้วเราจะให้ค่าตัวแปรอย่างใดก็ได้ดังแสดงในรูปที่ 3 (ขอให้ประเมินผลประโยคเหล่านี้ด้วย show it ทีละบรรทัด) ตัวแปรนี้จะคงอยู่ตลอดการรัน ST/V ครั้งนี้ (เว้นไว้จะเก็บเข้าระบบดังจะกล่าวในฉบับต่อไปครับ) อนึ่งเราสามารถจะกำหนดตัวแปรโกลบอลพร้อมกับการให้ค่าได้เช่นกัน

แต่สำหรับตัวแปรของคลาสที่ซับซ้อน เช่น คลาส pen (ที่ Turtle เป็นตัวแปรของคลาสนี้) จะต้องกำหนดโดยมีชื่อคลาสกำกับต่อด้วยคำว่า new เช่น

Abcd : = pen new

### ตัวแปรชั่วคราว (Temporary variable)

แต่ในการเขียนโปรแกรมโดยทั่วไปจะใช้ตัวแปรชั่วคราว (เทียบเท่ากับตัวแปรโลคอลในภาษาปาสคาล) ซึ่งจะคงอยู่เฉพาะในการรันโปรแกรมภาษาสมอลทอล์คครั้งหนึ่ง ๆ เท่านั้น การกำหนดตัวแปรแบบนี้ให้กำหนดไว้ในเครื่องหมายขีดตั้ง ดังรูปที่ 4

ขอให้ประเมินผลโปรแกรมทั้งหมดนี้ด้วย show it เครื่องจะแสดงค่าใน tem แต่ต่อไปถ้าถามค่าใน tem อีก ST/V จะตอบว่าไม่รู้จัก แตกต่างกับ Member ในรูปที่ 3 จะตอบค่าครั้งสุดท้าย (คือ 50) ให้ทราบเสมอ

```
#( 'array'  'of'  'four'  'string')
#( 'array' 'of' 3 'string')
#(('array of') ('array') ('of string'))
#(1 2 (3 4) ('and five'))
```

**รูปที่ 2 อาร์เรย์แบบต่าง ๆ**

```
Member
Member := 'Smalltalk'          'Smalltalk'
Member := 50                    50
Member                          50
```

**รูปที่ 3 ตัวแปรโกลบอล ด้านขวาคือผลของการประเมิน**

```
| tem num |
 tem := 30                          30
 tem
```

**รูปที่ 4 ตัวแปรชั่วคราว ด้านขวาคือผลของการประเมิน**

## สื่อความ (Messages)

ตามที่กล่าวว่าสื่อความคือคำสั่งที่เราสั่งการแก่ออปเจ็ค เป็นการมองโปรแกรมในแบบโอโอพี ซึ่งหากมองในแบบของโปรแกรมโดยทั่วไป สื่อความเทียบได้กับฟังก์ชัน (Function) ในภาษาปาสคาล แตกต่างกันตรงแทนที่จะนำออปเจ็คและสร้อยความคือ อาร์กิวเมนต์ (arguement) ไปรวมกันไว้ในวงเล็บ กลับนำออปเจ็คไปเขียนไว้หน้าสื่อความและเขียนอาร์กิวเมนต์ไว้หลังสื่อความ

การจำแนกสื่อความแยกได้ตามจำนวนอาร์กิวเมนต์และตามชนิดของสื่อความบางประเภท คือ

## สื่อความเดี่ยว (Unary Messages)

คือสื่อความที่ไม่มีอาร์กิวเมนต์ ดังแสดงในรูปที่ 5 (ซึ่งได้แสดงผลของการประเมินผลด้วย show it ไว้ในแต่ละบรรทัดด้วย) จะเห็นว่าคำที่ใช้เป็นสื่อความเช่น size จะขึ้นต้นด้วยอักษรเล็กเสมอ

## สื่อความผสม (Keyword Messages)

คือสื่อความที่ประกอบด้วยอาร์กิวเมนต์ตั้งแต่ 1 ตัวขึ้นไปดังแสดงในรูปที่ 6 ซึ่งแต่ละสื่อความจะปิดท้ายคำด้วยเครื่องหมาย : และสำหรับสื่อความที่มีหลายอาร์กิวเมนต์ คำที่ใช้เป็นสื่อความจะเป็นคำที่ใช้คู่กัน เช่น at : put : และ copyFrem : to : เป็นต้น

```
#('array' 'of' 'string') size          3
'now is the time' asUpperCase          'NOW IS THE TIME'
'hello there' reversed                 'ereht olleh'
#(4 'five' 6 7) reversed               (7 6 'five' 4)
$A asciiValue                          65
```

รูปที่ 5 สื่อความเดี่ยว ด้านขวาคือผลของการประเมิน

```
'now is the time' at: 6                          $s
'Hello' includes: $e                             true
'hello' at: 1 put: $H                            $H
'The quick brown' copyFrom: 4 to: 9              ' quick'
```

รูปที่ 6 สื่อความผสม ด้านขวาคือผลของการประเมิน

```
"Binary Messages"
"Evaluate each of the following with show it"
      5 + 7                          12
      'hello', ' there'              'hello there'
      #(1 2 3), #(4 5 6)             (1 2 3 4 5 6)
```

รูปที่ 7 สื่อความไบนารี ด้านขวาคือผลของการประเมิน

## สื่อความไบนารี (Binary Messages)

คือสื่อความที่ประกอบด้วยอาร์กิวเมนต์เพียงตัวเดียว กับสื่อความที่เรียกเป็น selector คือพวกเครื่องหมายต่าง ๆ แบบหนึ่งของสื่อความนี้คือ สื่อความคณิตศาสตร์ (Arithmetic Message) ดังแบบสื่อความไบนารีในรูปที่ 7 สื่อความที่เป็นเครื่องหมายในการคำนวณได้แก่

| | |
|---|---|
| + | บวก |
| – | ลบ |
| * | คูณ |
| / | หาร |
| // | หารแบบปัดเศษ |
| \\ | เศษของการหาร |

ซึ่งหมายถึงจะให้ผลของการบวก ลบ คูณ และต่อ ๆ ไปตามลำดับ สำหรับเครื่องหมาย , หมายถึง concatenates คือให้เชื่อมต่อ ตามตัวอย่างเป็นการเชื่อมต่อสตริงและอาร์เรย์ได้ผลดังที่แสดงที่ปลายบรรทัด

## สื่อความซ้อนสื่อความ (Messages inside of Messages)

ตัวอย่างง่าย ๆ ของการเขียนสื่อความซ้อนสื่อความ คือ การคำนวณดังแสดงที่บรรทัดแรกของรูปที่ 8 ซึ่งเมื่อเป็นสื่อความแบบเดียวกันในกรณีนี้คือไบนารีคณิตศาสตร์ สมอลทอล์คจะดำเนินการจากซ้ายไปขวาเสมอ (ไม่มีลำดับตามสำคัญของเครื่องหมาย)

แต่ถ้าเป็นสื่อความต่างแบบกันดังแสดงในบรรทัดต่อ ๆ ไป สมอลทอล์ค

```
"Messages inside of Messages"
"Evaluate each of the following with show it"
    1 + 2 - 3 * 4 / 5                          0
    'hello' size + 4                           9
    'now' size + #( 1 2 3 4 ) size             7
    #( 1 12 24 36) includes: 4 factorial       true
  4 factorial between: 3 + 4 and: 'hello' size * 7   true
```

**รูปที่ 8 สื่อความซ้อนสื่อความ ด้านขวาคือผลของการประเมิน**

```
"Expression Series"
"Evaluate the following as a single unit with do it"
        Turtle black.
        Turtle home.
        Turtle go: 100.
        Turtle turn: 120.
        Turtle go: 100.
        Turtle turn: 120.
        Turtle go: 100.
        Turtle turn: 120
```

**รูปที่ 9 ตัวอย่างโปรแกรม**

จะดำเนินการตามลำดับดังนี้คือ ทำสื่อความเดี่ยวก่อนไบนารี และทำไบนารีก่อนผสม เช่นตามรูป

- บรรทัดที่ 2 ทำ size ก่อน +
- บรรทัดที่ 3 ทำ size แรกก่อน แล้วทำ size หลัง แล้วจึงทำ +
- บรรทัดที่ 4 ทำ factorial ก่อน includes :
- บรรทัดที่ 5 จะทำตามลำดับดังนี้คือ factorial, size, +, * แล้วทำ between : and :

## รูปแบบของโปรแกรม

โปรแกรมในภาษาสมอลทอล์คไม่มีส่วนขึ้นต้นและลงท้าย (ดังเช่นในภาษาปาสคาล) หากประกอบด้วยประโยคซึ่งอาจจะมีเพียงประโยคเดียวหรือหลายประโยคก็ได้ หากมีหลายประโยคให้แยกจากกันด้วยเครื่องหมาย . นั่นคือประโยคสุดท้ายไม่จำเป็นต้องมีเครื่องหมายนี้ (เทียบได้กับที่ในภาษาปาสคาลใช้เครื่องหมาย ;) ดังเช่นโปรแกรมในรูปที่ 9 ซึ่งจะเป็นการสั่งเต่า (ตัวแปร Turtle) ให้เดินและจะปรากฏภาพเป็นรูปสามเหลี่ยมด้านเท่าบนจอภาพ

ลักษณะของโปรแกรมเต่า จะตรงกับแนวความคิดของการเขียนโปรแกรมในแบบโอโอพีที่สุด คือเป็นการสั่งออปเจ็คให้ดำเนินการตามที่เราต้องการ แตกต่างกับตัวอย่างก่อน ๆ ดังเช่นประโยค 3 + 5 ถึงแม้เราจะทำใจให้คิดว่าเป็นการสั่ง 3 ให้นำ 5 มาบวก แต่ในใจจริงคงมองเห็นว่า + เป็นฟังก์ชันที่จะให้ค่า 3 บวก 5 แก่เรา

## การรวมสื่อความ

ขอย้อนกลับไปที่รูปแบบของโปรแกรมครับ ตามโปรแกรมในรูปที่ 9 จะเห็นว่าทุกประโยคเป็นการสั่งการกับออปเจ็คเดียวกันคือ Turtle ในสมอลทอล์คจึงให้เราลดความซ้ำซ้อนได้ดังโปรแกรมในรูปที่ 10 คืออ้างถึงออปเจ็คครั้งเดียวแล้วแยกสื่อความด้วยเครื่องหมาย ,

## ประโยคในภาษาสมอลทอล์ค

ประโยคในสมอลทอล์คอาจเป็นเพียงนิพจน์ (Expression) เมื่อเทียบกับภาษาอื่น เช่น 5 * 10 แต่ถ้าเป็นโปรแกรมที่มีหลายประโยคจะต้องเป็นประโยคที่สมบูรณ์เทียบได้กับ statement ในภาษาปาสคาล เช่น a : = 5 * 10 เป็นต้น ซึ่งในภาษาสมอลทอล์คเรียกว่า Expression และมีอยู่ 4 แบบคือ

- ประโยคคำสั่ง เช่น 'hello' at : 1 put : $H
- ประโยคให้ค่า เช่น factorialTen : = 10 factorial
- ประโยคส่งค่ากลับ เช่น ^answer คือใช้เครื่องหมาย ^ นำหน้าตัวแปรเป็นคำสั่งให้ส่งค่าในตัวแปรกลับเมื่อจบการทำงาน
- ประโยคโครงสร้าง ซึ่งมีเรื่องที่จะต้องกล่าวถึงมากจึงขอยกไปเป็นหัวข้อใหญ่ครับ

## เรื่องของบล็อค (Block)

ในภาษาปาสคาลมีคำว่า begin end; เพื่อรวมถ้อยแถลงจำนวนหนึ่งไว้

ด้วยกัน เช่น ในการทำงานเป็นวงรอบเพื่อให้ทำถ้อยแถลงทั้งหมดนี้ ในสมอลทอล์คก็เช่นกันแต่ใช้เครื่องหมายวงเล็บก้ามปูเปิดและปิดในการกำหนดขอบเขตของบล็อคดังรูปที่ 11

[sum : = sum + n. n : = n+1]

[: n 1 sum : = sum + n]

บรรทัดแรกเป็นการรวมประโยคให้ค่า 2 ประโยคไว้ด้วยกัน สำหรับบรรทัดหลังเป็นการกำหนดตัวแปรชั่วคราว คือ n ไว้ใช้งานเฉพาะในบล็อคนี้เท่านั้น โดยให้ใช้เครื่องหมาย : นำหน้าชื่อตัวแปร และแยกจากส่วนอื่น ๆ ของบล็อคด้วยขีดตั้ง ซึ่งตัวแปรแบบนี้จะรับค่าจากสื่อความที่อยู่หน้าบล็อคด้วย ดังจะเห็นในตัวอย่างของประโยคโครงสร้าง

## นิพจน์เงื่อนไข

ก็เช่นเดียวกับทุกภาษาครับ เมื่อจะเขียนประโยคโครงสร้างก็จะต้องมีการตรวจสอบเงื่อนไข สำหรับในภาษาสมอลทอล์ค (เช่นเดียวกับในภาษาปาสคาล) สามารถจะตรวจสอบเงื่อนไขได้โดยสื่อความกับออปเจ็ค ซึ่งแบ่งได้เป็น

### สื่อความเปรียบเทียบ

สื่อความเปรียบเทียบคือเครื่องหมาย <,<=, =,>= และ > ซึ่งหมายถึง น้อยกว่า น้อยกว่าหรือเท่ากัน เท่ากัน มากกว่าหรือเท่ากัน และมากกว่าตามลำดับ กับไม่เท่ากัน ซึ่งผมเห็นว่าเรียงพิมพ์ยากจึงนำไปไว้ในตัวอย่างในรูปที่ 12

นิพจน์ทั้งหมดนี้จะให้ค่าความเป็นจริงทั้งสิ้น บรรทัดสุดท้ายคือสื่อความไม่เท่ากัน จะเห็นว่าเราสามารถเปรียบเทียบเลขจำนวนกับเลขจำนวน อาร์เรย์กับอาร์เรย์แบบเดียวกันและสตริงกับสตริงได้

### สื่อความให้ค่าความเป็นจริงหรือเท็จ

ดังที่กล่าวแล้วว่าสื่อความคือฟังก์ชัน มีสื่อความจำนวนหนึ่งที่ให้ค่าความเป็นจริงหรือเท็จ ซึ่งเราสามารถนำมาใช้เป็นเงื่อนไขได้ เช่น

7 odd

## นิพจน์บูลีน

ในกรณีที่เราต้องการเงื่อนไขที่เป็นผลจากนิพจน์เงื่อนไขมากกว่าหนึ่ง สามารถทำได้โดยใช้สื่อความ or : และ and : ดังรูปที่ 13

บรรทัดแรกเป็นการตรวจว่า C เป็นอักขระที่ไม่ใช่ตัวเลข (ใช่หรือไม่) บรรทัดที่สองตรวจว่า C เป็นอักขระที่เป็นตัวเลข (ใช่หรือไม่)

```
"Cascaded Messages"

"Evaluate the following with do it"

      Turtle
          white;
          home;
          go: 100;
          turn: 120;
          go: 100;
          turn: 120;
          go: 100;
          turn: 120
```

**รูปที่ 10 แสดงการรวมสื่อความไว้ในประโยคเดียวกัน**

```
[ sum := sum + n.  n := n + 1 ]
[ :n | sum := sum + n ]
```

**รูปที่ 11 บล็อค**

```
"Comparing Objects"

"Evaluate each of the following with show it"

      3 < 4

      5 <= (2 + 3)

      #( 1 2 3 4) = #(1 2 3 4)

      'hello' >= 'goodbye'

      $B > $A

      9 ~= 10
```

**รูปที่ 12 สื่อความเปรียบเทียบตามตัวอย่างนี้ให้ค่าความเป็นจริงทั้งหมด**

## ประโยคโครงสร้าง

ในภาษาสมอลทอล์คมีประโยคโครงสร้างเช่นเดียวกับในภาษาปาสคาล แต่มากกว่า กล่าวคือ

### การกระทำตามเงื่อนไข

กระทำเงื่อนไข IF-ELSE ดังแสดงในรูปที่ 14 ซึ่งมีการดำเนินการดังนี้

- บรรทัดแรกเป็นการกำหนดตัวแปรชั่วคราว 3 ตัวคือ max, a และ b
- ให้ค่า 5 ยกกำลังสองแก่ a และค่าแฟคทอเรียล 4 แก่ b

```
( c < $0 or: [c > $9])
( c >= $0 and: [c <= $9])
```

รูปที่ 13 สื่อความบูลีน

```
"Evaluate the following with show it"

    | max a b |
    a := 5 squared.
    b := 4 factorial.
    a < b
        ifTrue: [max := b]
        ifFalse: [max := a].
    ^max        25
```

รูปที่ 14 การกระทำตามเงื่อนไข ค่า 25 คือผลการประเมิน

```
"Evaluate the following with show it"

   Turtle
      black;
      home.
   3 timesRepeat: [ Turtle go: 100, turn: 120 ]
```

รูปที่ 15 วงรอบ timesRepeat

```
"Evaluate the following with show it"

    | string index c |
    string := 'Now is the time'.
    index := 1.
    string size timesRepeat: [
        c := string at: index.
        string
            at: index
            put:
                (c isVowel
                    ifTrue: [c asUpperCase]
                    ifFalse: [c asLowerCase]).
        index := index + 1].
    ^string                 'nOw Is thE tImE'
```

รูปที่ 16 แสดงการใช้สื่อความซ้อนสื่อความ

- นำมาเทียบกัน หากเป็นจริง (เทียบได้กับ IF) ให้ค่า b แก่ max หากเป็นเท็จ (เทียบได้กับ ELSE) ให้ค่า b แก่ max
- บรรทัดสุดท้ายเป็นการส่งค่าใน max กลับ ซึ่งยังไม่มีที่ใช้ในตอนนี้

จะเห็นว่าจากเงื่อนไขจนสุด if False เป็นประโยคเดียวกัน จึงไม่คั่นด้วยจุด เมื่อมองในแง่ของสมอลทอล์คทั้ง ifTrue : และ ifFalse เป็นฟังก์ชันที่ให้ค่าความเป็นจริง เมื่อ a น้อยกว่า b เป็นจริงหรือเป็นเท็จตามลำดับ เมื่อเป็นจริงก็จะดำเนินการในบล็อคที่ตามมาได้ผลการประเมินผลดังแสดงในรูป

### วงรอบ (timesRepeat :)

คำว่า timeRepeat : เป็นสื่อความที่ทำให้เกิดการทำงานเป็นวงรอบตามจำนวนครั้งของออปเจ็ค (เทียบได้กับวงรอบ FOR ในภาษาปาสคาล) ดังโปรแกรมในรูปที่ 15 โปรแกรมนี้จะทำงานในบล็อคซ้ำ 3 ครั้ง ปรากฏภาพเป็นรูปสามเหลี่ยมด้านเท่าบนจอภาพ

เพื่อให้เห็นการทำงานที่ซับซ้อนขึ้น ขอยกตัวอย่างโปรแกรมดังรูปที่ 16 โปรแกรมนี้จะแปลงอักษรเล็กที่เป็นสระในสตริง Now is the time ให้เป็นอักษรใหญ่ ดังผลการรันที่บรรทัดสุดท้าย โปรแกรมนี้ประกอบด้วยประโยคใหญ่ 3 ประโยคคือ ที่ให้ค่าสตริง ให้ค่า 1 แก่ index ที่เหลือคือประโยคที่ 3 ที่มีการใช้สื่อความซ้อนสื่อความมากพอสมควร กล่าวคือ

- size จะให้ค่าความยาวของสตริง ซึ่งจะเป็นจำนวนครั้งของการทำงานในวงรอบ timesRepeat :
- ในบล็อค ประกอบด้วยประโยค 3 ประโยค คือ ประโยคแรกเป็นการให้ค่าอักขระในสตริงตรง index ประโยคที่สามคือประโยคสุดท้ายที่เพิ่มค่าอินเด็กซ์ขึ้นหนึ่ง

- ที่เหลือคือประโยคที่ 2 นำอักขระไปเก็บในสตริง โดยให้แปลงเป็นอักษรใหญ่ก่อน ถ้าเป็นสระ จึงให้การทำงานในส่วนนี้อยู่ในวงเล็บ ( )

### วงรอบ to : by : do :

เป็นการทำงานเป็นวงรอบที่สามารถกำหนดค่าเริ่มต้นและสุดท้ายได้คือ to : และหากต้องการกำหนดขนาดของการเพิ่มค่าให้กำหนดด้วย by : ดังตัวอย่างในรูปที่ 17

โปรแกรมแรกเป็นการให้ค่า 3 ถึง 6 แก่ i โดยเพิ่มค่าครั้งละ 1 (เทียบได้กับวงรอบ FOR ในภาษาปาสคาลได้โดยตรง) สำหรับโปรแกรมหลังเป็นการให้ค่าตั้งแต่ $\frac{1}{2}$ ถึง 1 แก่ i เช่นกัน แต่เพิ่มค่าขึ้นครั้งละ $\frac{1}{8}$

### วงรอบ WHILE

ในสมอลทอล์คมีการทำงานเป็นวงรอบในแบบ WHILE อยู่ 2 แบบ คือ whileTrue กับ whileFalse เพื่อใช้กำหนดเงื่อนไขในการออกจากวงรอบ คือให้คงทำงานในวงรอบเมื่อเงื่อนไขเป็นจริงและเป็นเท็จตามลำดับ (ทั้งคู่นี้เทียบได้กับวงรอบ WHILE ในภาษาปาสคาล) ดังโปรแกรมตัวอย่างในรูปที่ 18

โปรแกรมแรกเป็นการนับจำนวนสระในสตริงโดยตั้งเงื่อนไขไว้ว่า ถ้าค่าใน index น้อยกว่าหรือเท่ากันกับความยาวของสตริง ให้คงทำงานในวงรอบ

โปรแกรมหลังเป็นการลอกข้อความจากไฟล์ go (ซึ่งเป็นไฟล์ที่มีอยู่แล้ว) ให้แก่ไฟล์ junk (ซึ่งเป็นไฟล์ใหม่ โดยตั้งเงื่อนไขไว้ว่าถ้ายังไม่สุดไฟล์ input ให้คงทำงานในวงรอบ

## สื่อความที่ทำงานเป็นวงรอบ

ความจริงสื่อความที่กล่าวมาต่างก็เป็นสื่อความที่ทำงานเป็นวงรอบ แต่ใน

```
"Evaluate the following with show it"

    | sum |
    sum := 0.
    3 to: 6 do:  [ :i |
       sum := sum + i.
    ^sum            18


    "compute the sum of 1/2, 5/8, 3/4, 7/8, 1"
    | sum |
    sum := 0.
    1/2 to: 1 by: 1/8 do: [ :i |
         sum := sum + i].
    ^sum            15/4
```

**รูปที่ 17 วงรอบ to : by : do :**

```
"Evaluate the following with show it"

    "count vowels in a string"
    | vowels string index |
    vowels := 0.
    index := 1.
    string := 'Now is the time'.
    [index <= string size]
        whileTrue: [
            (string at: index) isVowel
                ifTrue: [vowels := vowels + 1].
            index := index + 1].
    ^vowels          5


"Evaluate the following with do it"

    "copy a disk file"
    | input output |
    input := File pathName: 'go'.
    output := File pathName: 'junk'.
    [input atEnd]
         whileFalse: [output nextPut: input next].
    input close.
    output close
```

**รูปที่ 18 แสดงการใช้วงรอบ WHILE สองแบบ**

```
"Evaluate the following with show it"

    | sum |
    sum := 0.
    #( 1 2 3 4 5) do: [ :num |
        sum := sum + num ].
    ^sum         15


"Evaluate the following with show it"

    "count vowels in a string"
    | vowels |
    vowels := 0.
    'Now is the time' do: [ :char |
        char isVowel
            ifTrue: [vowels := vowels + 1]].
    ^vowels          5
```

**รูปที่ 19 แสดงการใช้วงรอบ do :**

สมอลทอล์คมีสื่อความที่ทำงานเป็นวงรอบพิเศษไปอีกแบบหนึ่ง และต้องใช้คู่กับการกำหนดตัวแปรชั่วคราวในบล็อค กล่าวคือ

*** วงรอบ do:**

เราได้ใช้ do : มาบ้างแล้วกับ to : โปรแกรมตัวอย่างในรูปที่ 19 เป็นการใช้งานในอีกลักษณะหนึ่งคือ do: จะให้ค่าอาร์เรย์ทีละจำนวน หรือค่าอักขระในสตริงทีละตัวแก่ตัวแปรโลคอลในบล็อค ตามโปรแกรมแรกคือ : num ส่วนโปรแกรมหลังคือ : char

ส่วนในบล็อค สุดแต่จะดำเนินการใด เช่นในโปรแกรมแรกให้หาผลบวกของเลขในอาร์เรย์ ส่วนโปรแกรมหลังให้นับจำนวนสระในสตริง (เช่นเดียวกับโปรแกรมในรูปที่ 18)

```
"Evaluate the following with show it"

"The select: Iterator"

    'Now is the time' select: [ :c | c isVowel ]
                'oieie'

"The reject: Iterator"

    "answer all digits whose factorial is
    less than the digit raised to the 4th power"
    #( 1 2 3 4 5 6 7 8 9 ) reject: [ :i |
        i factorial >= ( i * i * i * i ) ]

            (2 3 4 5 6)

"The collect: Iterator"

    "square each element in the array"
    #(1 13 7 10) collect: [:i| i * i]

                (1 169 49 100)
```

**รูปที่ 20 วงรอบ select :, reject : และ collect :**

```
"Evaluate the following with show it"

    "compute the value of the first integer in a string"
    | string index answer c |
    string := '1234 is the number'.
    answer := 0.
    index := 1.
    string size timesRepeat: [
        c := string at: index.
        (c < $0 or: [c > $9])
            ifTrue: [^answer].
        answer := answer * 10
            + c asciiValue - $0 asciiValue.
        index := index + 1].
^answer         1234
```

**รูปที่ 21 แสดงการส่งค่ากลับเพื่อออกจากวงรอบ**

*** วงรอบ select :**

จะทำงานตามเงื่อนไขของอาร์กิวเมนต์คือในบล็อค และให้ผลเป็นผลรวม ดังเช่น โปรแกรมแรกในรูปที่ 20 จะให้ผลเป็นสตริงที่ประกอบด้วยสระที่อยู่ในสตริงที่เป็นออปเจ็ค

*** วงรอบ reject :**

ดำเนินการเช่นเดียวกันกับ select: แต่ตรงข้ามกันคือ จะให้ค่าที่เงื่อนไขในบล็อคเป็นเท็จ เช่นตามตัวอย่างของ select : หากเปลี่ยนเป็นใช้ reject แทน ค่าสตริงที่ได้จะเป็นสตริงของพยัญชนะ (ที่ไม่มีสระ)

หรือตามโปรแกรมที่ 2 ในรูปที่ 20 จะให้ค่าเป็นอาร์เรย์ของเลขจำนวนที่มีค่าแฟคทอเรียลน้อยกว่าค่ายกกำลัง 4

*** วงรอบ collect :**

มีการทำงานเช่นเดียวกับ do: (คือไม่เกี่ยวกับเงื่อนไขในบล็อคดัง select: หรือ reject:) แต่จะให้ผลรวมของการทำงานในบล็อค ดังโปรแกรมที่ 3 ในรูปที่ 20

## การออกจากวงรอบ

ในงานบางงานเราอาจต้องการออกจากวงรอบเมื่อได้เงื่อนไขที่ต้องการ โดยไม่ต้องทำงานจนครบวงรอบ (ดังเช่นในภาษาปาสคาลใช้ Exit) จะดำเนินการได้ด้วยการส่งค่ากลับ ดังโปรแกรมตัวอย่างในรูปที่ 21

โปรแกรมนี้จะแปลงแถวของตัวเลขที่อยู่ต้นสตริง เป็นเลขจำนวน เงื่อนไขที่ใช้ออกจากวงรอบคือ เมื่ออักขระในสตริงไม่ใช่ตัวเลข จึงให้ส่งค่า answer กลับเป็นการออกจากวงรอบ

เรื่องราวของออปเจ็คกับสื่อความที่สำคัญคงมีเพียงเท่านี้ครับ ในฉบับหน้าจะได้นำเรื่องราวของสกุลกับกิจกรรมมากล่าวถึงต่อไป